Turn Based Communication Channel

Carlo Brunetta[1], Mario Larangeira[2], Bei Liang[3], Aikaterini Mitrokotsa[1] and Keisuke Tanaka[2]

[1] Chalmers University of Technology, Gothenburg, Sweden
[2] Department of Mathematical and Computing Sciences, School of Computing, Tokyo Institute of Technology, Tokyo, Japan
[3] Beijing Institute of Mathematical Sciences and Applications, Beijing, China

Under Submission

Abstract: We introduce the concept of turn-based communication channel between two mutually distrustful parties with communication consistency, *i.e.* both parties have the same message history, and happens in sets of exchanged messages across a limited number of turns. Our construction leverages on *timed primitives*. Namely, we introduce a novel Δ-delay hash function definition in order to establish *turns* in the channel. Concretely, we introduce the one-way turn-based communication scheme and the two-way turn-based communication protocol and provide a concrete instantiation that achieves communication consistency.

Keywords: Time Puzzle, Delay, Hash Function, Consistency

1 Introduction

Communication channels are the core mediums allowing different parties to build dialogues. They can either be *physical* or *abstract*, *e.g.* electromagnetic wave propagation or a key exchange protocol that allows to *establish a secure communication channel*. Either the case, channels achieve different properties which can be related to the medium, *e.g.* reliability, energy efficiency, bandwidth, or based on the *"content"*, *e.g.* confidentiality, privacy or other. A fundamental and highly desirable property of a channel is *consistency*, *i.e.* different parties exchange messages which cannot be modified or repudiated in the future once the communication is over. In other words, whenever a message is shared, it is permanently fixed in the transcription. An example of a protocol that allows such a property is the *public bulletin board* which allows any party to publish any information on the *"board"*, while receiving a *"proof"* that guarantees the *integrity* that the information is indeed *published*. Recently, blockchains, or public ledgers [BGM16, KRDO17], have emerged as complex protocols that allow the instantiation of a public bulletin board, without relying on a central authority.

Their security relies on a specially purposed *consensus protocol*, which often requires assumptions of game-theoretic nature, *e.g.* the *proof-of-work* consensus protocol implies that an adversary does not have more than 51% of the available computing power at its disposal. Bulletin boards based on consensus protocols, albeit practical, suffer from significant delays when persisting entries. Notably, blockchain-based systems, typically suffer from scalability issues without a clear solution yet. Consequently, for time critical systems, blockchain-based bulletin boards may not be a useful alternative. An emerging technology, autonomous driving, illustrates the challenge between time-critical systems and blockchains. Autonomous driving in a real-world environment is a notoriously hard task because of the high number of variables that must be taken into account. Moreover, in such systems, communication between cars is a viable design approach. Different systems must communicate and coherently agree on their action plans.

Let us consider a simplified example where a car is overtaking another one. The one taking the action and surrounding cars must securely execute their algorithms while communicating to each other. All the communication between the cars should be timely available and guaranteed to be correct, *i.e.* could not be changed a posteriori, for audit purposes. The transcript of the whole communication could be used later, or even in court, for legal issues. A straightforward approach is to let vehicles be equipped with cryptographic primitives, such as digital signatures. Despite its feasibility, the aid of public key cryptography may not be an option in for some devices, in particular, resource restricted ones. Besides, it may require the use of Public Key Infrastructure (PKI) which may be, again, prohibitive for some systems.

One of the most basic building blocks in cryptographic literature are *hash functions*. They are used to guarantee data integrity and are widely employed in the computer science discipline in numerous applications. A natural question is whether such a building block would allow the construction of a *pair-wise communication channel*, avoiding the somewhat heavier cryptographic primitives earlier cited. An application relying only on hash functions could be significantly *"easier"*, since it would not be aided by public key cryptography schemes with PKI, typically more *"complex"* than their private key cryptography counterpart. Furthermore, it could also sidestep the early mentioned limitations of blockchain based protocols, yet providing a consistent and timely communication channel between two users. More succinctly, we investigate the following question:

is it possible to design a consistent channel between two parties ***without*** *using blockchain's assumptions* ***nor*** *public key infrastructure?*

Next, we detail the main approach of our idea which is to devise a *"turn"*, such that messages are exchanged only within the turns, and the proofs of submitted messages, similary to a bulletin board, are generated in order to guarantee consistency. The set of all turns of the channel, *i.e.* it contains a finite number of them are purposely related to each other. Therefore, they are not easily altered without affecting the overall transcript proofs of the exchanged communication.

Concept's Overview. All the communication is held over *time* which allows to *order* events during communication, *e.g.* message exchange. Commonly, our daily interaction is held over **continuous communication channels** in which the communicating parties can communicate at *any* point in time.

Our main idea, as depicted in Fig. 54, relies on providing a **turn-based communication channel (TBCC)** that forces the two parties to communicate in a *limited* amount of distinct *turns* separated by a Δ time interval. The interaction between the parties is slowed down by the necessity of *waiting for the next turn*, contrary to the almost-instantaneous reply ability of continuous channels.

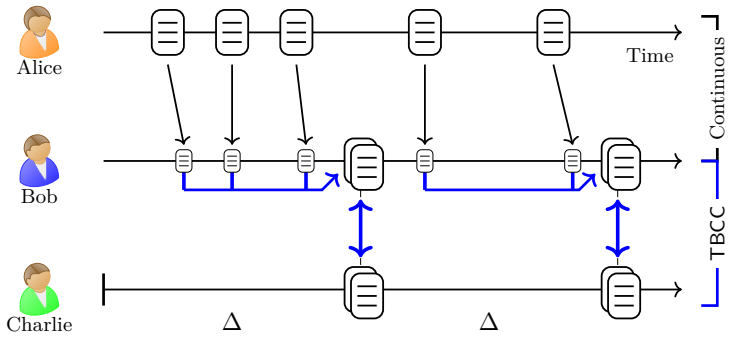

Figure 54: A continuous and TBCC channel, the messages are gathered in "blocks", and each block, and its set of messages, is confirmed only at the end of each turn.

To do so, we assume the existence of functions that *"computationally"* create time delays and are used to extend the hash function definition and introduce the Δ-delay hash function, which paves the way to the construction of **time-lock puzzles** in the spirit of Mahmoody *et al.* [MMV11], *i.e.* a primitive that allows Alice P_A to generate a puzzle-solution pair (y, π), send the puzzle y to Bob P_B that spends a time Δ to compute the solution π. Concretely, Δ is the turn interval in our TBCC construction. The novel feature provided by TBCC is that P_A knows the solution π in advance and can use it to *"commit"* to a message m. By releasing m and the puzzle y, P_B must invest Δ amount of time in computing π *before being able* to verify the validity of m. The early described *timed-commitment* is the stepping stone of our first construction for a **one-way turn-based scheme** that allows the communication of blocks of messages in turns in a single direction, *e.g.* from P_A to P_B. We show that if the one-way turn-based scheme is correct and tamper resistant, *i.e.* the adversary is unable to modify the past communication and/or the correctness of the exchanged messages, intuitively this yields to **communication consistency**, *i.e.* both parties have the same view of the exchanged messages even if the adversary delays/tampers any message. We define the **two-way TBCC protocol** as a *"two one-way scheme"* which allows a simpler extension of the properties to the protocol, *i.e.* correctness, tamper resistance, sequentiality and consistency. Additionally, we introduce the concept of **turn synchronisation**,

i.e. the two communicating parties must always agree in which *shared* turn they are communicating. The protocol can further provide a **recovery procedure** that allows the communicating parties to fix the last-turn messages in case of a communication error or an adversarial tamper. We summarise our ideas and contributions in Fig. 55.

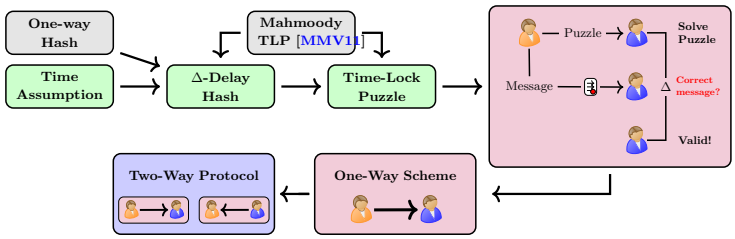

Figure 55: Roadmap of our contributions where we depict in gray the common assumption and definitions, in green our assumptions and basic primitives, in purple our main idea and construction and in blue our main contributions.

1.1 Related Work

Blockchains and Bulletin Boards. The blockchain data structure is commonly used in a distributed environment, where cryptographic primitives intersect with game theoretical assumptions in order to create a distributed database, where consistency comes for the orderly generation of blocks added to the structure. In the literature there are many examples of either *using* blockchains as a building block with new primitives, *e.g.* public verifiable proofs [SSV19], or applying existing cryptographic primitives into blockchains in order to achieve new functionalities [BBF19, KMS14]. Other focus is dedicated to the theoretical aspects related to the consensus mechanism or the blockchains' theoretical model [GKL15].

Time and Cryptographic Primitives. Cryptography and timing are long time distinct aspects that are commonly not considered together. Rivest *et al.* [RSW96] described the possibility of using time to create a *cryptographic time-capsule*, *i.e.* a ciphertext that will be possible to decrypt after a specified amount of time. Their work defines the concept of *time-lock puzzles*, where timing is achieved by cleverly tweaking the security parameters of some secure cryptographic primitives, *e.g.* choose a specific parameter λ such that the computational complexity of a specific problem is solvable by a real machine in reasonable time. Boneh *et al.* [BN00] presented the concept of timed commitments, *i.e.* a commitment scheme in which at any point, by investing an amount of effort, it is possible to correctly decommit into the original message. The main conceptual difference with respect to previous works is that, in this work, timing properties are achieved by forcing the algorithm to compute a naturally sequential mathematical problem. From a different perspective, Mahmoody *et al.* [MMV11] defined time-lock puzzles by just assuming the existence of timed primitives.

In the last years, many community efforts are spent into the definition of *verifiable delay functions* (VDFs), *i.e.* to compute a timed function and be able to verify the correct computation of it. There are multiple instantiations of this primitive in the literature, *e.g.* Lenstra *et al.*'s random zoo [LW15], a construction using randomized encoding by Bitansky *et al.* [BGJ+16] or Alwen-Tackmann's theoretical consideration regarding *moderately hard functions* [AT17]. The VDF's formal definition is given by Boneh *et al.* [BBBF18], subsequent papers provide additional properties for these time

related primitives such as Malavolta-Thyagarajan's homomorphic time-puzzles [MT19] or the *down-to-earth* VDF instantiation by Wesolowski [Wes19].

Timing Model. Perhaps the closest set of works to our study deals with the Timing Model as introduced by Dwork *et al.* [DNS04], and used by Kalai *et al.* [KLP07]. While they do present similarities to our work, *e.g.* the idea of *"individual clock"*, they also present significant differences. For instance, while in [DNS04, KLP07] every party in the real execution is equipped with a *"clock tape"*, extending the Interactive Turing Machine (ITM) with clocks, in our model the parties are regular ITMs, that perform computations in order to realize a *"single clock"* used by the ideal functionality. Additionally, our work also shares similarities with Azar *et al.* [AGP16] work on *ordered MPC*, which studies delays and ordered messages in the context of MPC. Our framework is positioned between both models as it focuses on turns equipped with a message validating mechanism, which is a different approach.

Recently, a concurrent and theoretical work by Baum *et al.* [BDD+20] formalizes the security of time-lock puzzles in the UC framework. More concretely they introduce the *UC with Relative Time* (RUC), which allows modelling relative delays in communication and sequential computation without requiring parties to keep track of a clock, in contrast to Katz *et al.*'s [KMTZ13] approach which models a *"central clock"* that all parties have access. The main contribution introduces a *semi-synchronous* message transmission functionality in which the adversary is aware of a delay Δ used to schedule the message exchanges, while the honest parties are not aware. In their work, composable time-puzzle realizes such novel functionality, and yields UC secure fair coin flips and two party computation achieving the notion of *output independent abort*. They focused on composable primitives and therefore have to rely on a constrained environment, *i.e.* it has to signal the adversary and activate every party at least once. Another theoretical difference is the focus of the order and turns but not in relative delays as in [BDD+20].

Baum *et al.* state as future work a possible extension to their transmission model in which all the parties have a *local clock* that would allow to always terminate any protocol. Our paper tackles that extension and provides a tangible instantiation of the extended model.

Paper Organisation. Sec. 2 states the preliminaries and time-complexity assumption. Sec. 3 defines the one-way and two-way TBCC protocol and related properties. Sec. 4 presents a collectively flip-coin protocol between two parties.

2 Preliminaries

In this section, we present notations and assumptions used throughout the paper.

We denote vectors with bold font, *e.g.* $\mathbf{v}$, and $\mathsf{Pr}[E]$ the probability of the event E. Let $\{0,1\}^*$ be the binary strings space of arbitrary length, $\mathbb{N}$ the natural numbers, $\mathbb{R}$ the real numbers and $\mathbb{R}_+$ the positive ones. Let $[a,b]$ denote intervals between a and b and $x \leftarrow_R X$ the random uniform sampling in the set X. Let $\mathsf{negl}(\lambda)$ denote a negligible function in λ, *i.e.* $\mathsf{negl}(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$. We omit λ whenever obvious by the context.

Definition 40 (One-Way Hash Function [KL08])**.** *Let $n \in \mathbb{N}$. The function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ is a **one-way hash function** if it satisfies the properties:*

- ***Preimage resistance:** for any $x \leftarrow_R \{0,1\}^*$ and $y := \mathsf{H}(x)$, for any PPT adversary $\mathcal{A}$ that, on input y, outputs x', it holds that $\mathsf{Pr}[\mathsf{H}(x') = y] < \mathsf{negl}$;*
- ***2nd Preimage resistance:** for any $x \leftarrow_R \{0,1\}^*$, $y := \mathsf{H}(x)$, for any PPT adversary $\mathcal{A}$ that, on input x, outputs $x' \neq x$, it holds $\mathsf{Pr}[\mathsf{H}(x') = y] < \mathsf{negl}$;*

Complexity and Time. Let time be modelled as the positive real numbers $\mathbb{R}_+$. At the core of our construction, we must assume the existence of a measure $\mu(\cdot)$ that plays the role of a *"bridge"* between *complexity and timing*. Formally,

Assumption 7. *Given a model of computation $\mathcal{M}$, there exists a measure $\mu(\cdot)$ that takes as input an $\mathcal{M}$-computable function f with input x and outputs the amount of time $\mu(\mathsf{f}, x) \in \mathbb{R}_+$ necessary to compute $\mathsf{f}(x)$ in the model $\mathcal{M}$. If $\mathsf{f}^\star(x)$ is a* probabilistic *function with input x and internal randomness r, then there exists $\mathsf{f}(x; r)$* deterministic *function that executes $\mathsf{f}^\star(x)$ with fixed randomness r.*

Informally, given a model of computation, *e.g.* Turing machines, quantum computers, *"pen-and-paper"*, it is possible to measure *"how much time does it take"* to compute $\mathsf{f}(x)$ both in the cases when f is deterministic or probabilistic[14].

Another required assumption is the existence of a function family $\mathcal{F}$ of which functions always output the results after the same amount of time. Formally,

Assumption 8. *Given a model of computation $\mathcal{M}$ and associated $\mu(\cdot)$, there exists a function family $\mathcal{F}$ such that for any function $\mathsf{f} \in \mathcal{F}$, for any inputs x, x', f is* input-independent *with computing time $\mu(\mathsf{f})$,* i.e. *$\mu(\mathsf{f}) = \mu(\mathsf{f}, x) = \mu(\mathsf{f}, x')$.*

Through the remaining of this work, we consider timing as the output of $\mu(\cdot)$ applied on input-independent functions. Whenever not specified, a *hard* problem is a problem of which solution, computed via f, has *large* computation time $\mu(\mathsf{f})$.

The *timed* one-way hash function extends the hash's properties of Def. 40.

Definition 41 (Δ-Delay One-Way Hash Function)**.** *Let $n \in \mathbb{N}$. The function $\overline{H} : \{0,1\}^* \to \{0,1\}^n$ is a* ***Δ-delay one-way hash function*** *if it is input-independent as described in Assumption 8 and, in addition to the properties of Def. 40, the following property also holds:*

- ***Δ-Delay:*** *for any PPT adversary $\mathcal{A}$ that takes an input x and outputs y which runs in time $\mu(\mathcal{A}, x) < \Delta = \mu(\overline{H})$, it holds that $Pr[y = \overline{H}(x)] < \mathsf{negl}$.*

Observe that, in order for the Δ-delay's property to make sense, the length of x might require to be limited, *e.g.* x must be polynomial. We omit such detail and always consider delay hash functions with the appropriate input space size.

Define the **time-lock puzzle** (TLP) as a *generate-solve* algorithm pair in which time plays a design/security aspect. Our definition is inspired by Azar *et al.* [AGP16] and, more specifically, we consider the construction presented by Mahmoody *et al.*'s [MMV11] in the random oracle (RO) model. The provided TLP generates $m+1$ *sequential* puzzles, *i.e.* a list of **partial puzzle** y_i of which **partial solution** π_i is necessary in order to solve the next partial puzzle y_{i+1}.

Definition 42 (Time-Lock Puzzle)**.** *Let $m \in \mathbb{N}$, security parameter λ and $\Delta \in \mathbb{R}_+$ be the desired time delay. Let $\overline{H} : \{0,1\}^* \to \{0,1\}^n$ be a Δ-delay hash function for some $n \in \mathbb{N}$. Let the algorithms $(\mathsf{GenPuz}, \mathsf{SolPuz})$ define a $(m\Delta)$* ***time-lock puzzle*** *$(m\Delta$-TLP$)$ as:*

- *$\mathsf{GenPuz}(\lambda, (m, \Delta)) \to (\mathbf{y}, \pi)$: the generation algorithm randomly samples $m+1$ bit-strings $x_i \in \{0,1\}^n$ and it computes the hash $\overline{H}(x_i)$ for $i \in [0, m]$. The algorithm outputs the list of partial puzzles and partial solutions:*

$$(\mathbf{y}, \pi) \coloneqq \Big(\big(x_0, \overline{H}(x_0) \oplus x_1, \dots, \overline{H}(x_{m-1}) \oplus x_m\big), (x_0, x_1, \dots, x_m) \Big);$$

[14] Observe that the *same* computational problem might have *different* timing, *e.g.* solving a classic-secure discrete logarithm instance is infeasible on a classical computer while it is theoretically feasible on a quantum computer.

- $\mathsf{SolPuz}(\mathbf{y}, k, (\pi_0, \dots, \pi_{k-1})) \rightarrow \pi_k$: *the algorithm parses* $\mathbf{y}$ *into* $(y_0, y_1, \cdots, y_m)$, $k \in [1, m]$ *and the known partial solutions* $(\pi_0, \dots, \pi_k)$. *It then outputs the partial solution* $\pi_k \coloneqq y_k \oplus \overline{H}(\pi_{k-1})$ *where* $\pi_0 \coloneqq y_0$.

The following three properties must hold:

- ***Correctness****: for every delay* Δ, *security parameter* λ *and* $m, n \in \mathbb{N}$, *for every puzzle* $(\mathbf{y}, \pi) \leftarrow \mathsf{GenPuz}(\lambda, (m, \Delta))$, *for every* $k \in [1, m]$, *it holds that*

$$Pr[\mathsf{SolPuz}(\mathbf{y}, k, (\pi_0, \dots, \pi_{k-1}))=\pi_k] = 1$$

- ***Timing****: for every delay* Δ, *security parameter* λ *and values* $m, n \in \mathbb{N}$, *for every puzzle* $(\mathbf{y}, \pi) \leftarrow \mathsf{GenPuz}(\lambda, (m, \Delta))$, *for every* $k \in [1, m]$ *it holds that* $\mu(\mathsf{SolPuz}) = \Delta$ *and generating the puzzle is faster than solving it,* i.e.

$$\mu(\mathsf{GenPuz}) \leq m \cdot \mu(\mathsf{SolPuz})$$

- ***Locking****: for every delay* Δ, *security parameter* λ *and values* $m, n \in \mathbb{N}$, *for every puzzle* $(\mathbf{y}, \pi) \leftarrow \mathsf{GenPuz}(\lambda, (m, \Delta))$, *for every* $k \in [1, m]$ *and adversary* $\mathcal{A}$ *that solves the* k*-th partial puzzle,* i.e. $\mathcal{A}(y, k, (\pi_0, \dots, \pi_{k-1})) = \pi_k$, *it holds that* $\mu(\mathcal{A}) < \Delta$ *with only negligible probability.*

The $(m\Delta)$-TLP describes a sequence of sequential puzzles that must be solved one at a time. The timing property guarantees that the SolPuz algorithm requires a specific Δ amount of time to be executed and that generating the whole puzzle takes less time than solving all the m puzzles. The locking property guarantees that any adversary $\mathcal{A}$ is unable to solve the partial puzzle in less time than Δ which implies, intuitively, that SolPuz is the *most optimised* algorithm for solving the partial puzzle y_i. If a better solving algorithm SolPuz' exists with solving time $\Delta' < \Delta$, then $(\mathsf{GenPuz}, \mathsf{SolPuz}')$ is a $(m\Delta')$-TLP while $(\mathsf{GenPuz}, \mathsf{SolPuz})$ cannot satisfy the locking property.

3 Instantiating the Turn Based Communication Channel

In this section, we discuss the core concepts of **timed disclosure**, **turns block** and **communication consistency**, later used to fully instantiate one and two-way TBCC, from a time-lock puzzle based on a Δ-delay hash function.

Timed Disclosure and Message Block. Consider a Δ-delay hash function and the related time-lock puzzle (y, π) as defined in Def. 42. Alice generates and publishes the puzzle y. On receiving y, Bob starts solving it. Within the amount of time Δ, only Alice knows the solution π, which allows her to produce an efficient digest $\xi = \mathsf{H}(\mathsf{m}, \pi)$ for any message m that she wants to communicate with Bob. At this stage, Bob is unable to compute the same digest because he does not know π. The *"timed disclosure"* is achieved whenever Bob finds the solution π which enables him to accept or reject the previously received message by verifying the correctness of the digest ξ. *Timing is key* for the security of the disclosure: Alice must use the knowledge **before** it is disclosed and, on the other hand, Bob should reject anything that uses such secret **after** the disclosure. Differently, **only** after Δ time, Bob can check which are the correct messages that are blinded to the specific solution π and can collect them into a **turn block**. Whenever we consider that Alice can publish a sequential time-lock puzzle in which one partial solution π_i is the *starting point* for the next partial puzzle y_{i+1}, Bob must filter and accept the received messages into a block every Δ amount of time therefore creating the concept of **turns** and relative message blocks. This **turn point-of-view** is possible because of the *sequential timed disclosure* that can be seen as a *"clock that ticks"* every Δ amount of time. This means that the communication is

one-way, from Alice to Bob. Alice does *not see* the turn because all the partial solutions are known to her and therefore she is able to generate any possible message-digest pair at any time, see Fig. 56.

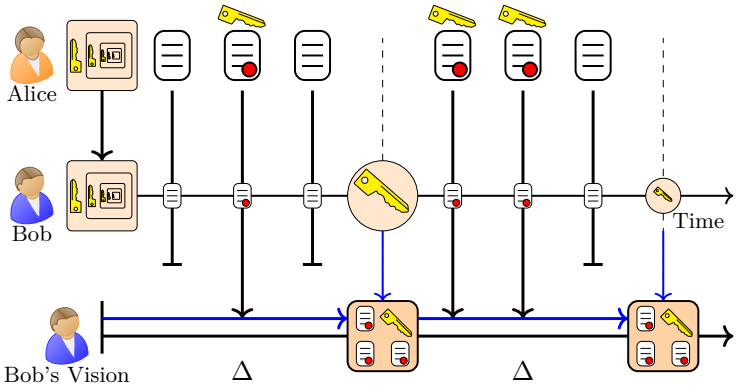

Figure 56: One-way channel scheme representation. Alice shares a time-lock puzzle with Bob and then sends messages of which some are correctly binded with the next puzzle's partial solution. With that solution, Bob is able to filter out the correct messages. Since this is done every Δ time, in Bob's eyes is as if he is receiving messages in turns.

Block of Messages and Communication Consistency. The next step is to create a *two-way* communication between Alice and Bob by allowing them to instantiate two independent one-way TBCC channels between each other, *i.e.* by exchanging time-lock puzzles and communicating message-digest pairs that are accepted and personally saved in blocks. These blocks are not stored in a trusted third party service but Alice and Bob have their own local copy of the exchanged message history and this means that it is required to provide a procedure to guarantee **consistency** between the copies. Consider our communicating Alice and Bob to be in the i-th turn, *i.e.* at the end of the turn they will create the i-th block. Naively, to achieve consistency of all blocks, every message, of the current block, should be bound to the *previous and future block*. For the *previous block*, they include a digest h_{i-1} of the previous block in every message they share in order to correctly verify that both have the same previous block vision. When the i-th turn ends, they separately create their own block-vision which could be different. When they enter the $(i+1)$-th turn, they will have to share the previous block digest h_i and they will see that the values are different. They will therefore start a **recovery phase** by publishing the content of the i-th block. At this point in time, the message's digest ξ_i can be tampered by anyone since the partial solution π_i is publicly known. For this reason, for every message we define a second digest σ_i that binds such message with the *next turn/future block* solution π_{i+1}. This procedure allows every party to understand *"who is cheating"* or *"where the errors are"*. In this way it is possible to abort the communication at any point in time, whenever a malicious party hijacks the channel. All the parties are thus forced to honestly participate if they want to maintain the channel up.

3.1 One-Way TBCC Instantiation

In this section, we instantiate the turn-based one-way channel from Alice to Bob. A *"channel"* is any collection of parameters that allows to participate into the communic-

ation, *e.g.* whenever a list of parameters is published, anyone can use them to correctly parse future messages shared using them.

Definition 43. *The* ***one-way channel scheme*** *is defined with the PPT algorithms* ($\mathsf{setup}, \mathsf{send}, \mathsf{ext}, \mathsf{turntoken}, \mathsf{valid\text{-}ver}, \mathsf{tamper\text{-}ver}$) *as:*

- $\mathsf{setup}(\lambda, \Delta, n) \rightarrow (\mathcal{C}, \mathcal{C}_{priv})$*: to setup the communication channel,* P_A *parses the security parameter* λ*, the delay* Δ *and the number of turns* n *The setup algorithm outputs the public and private channels* $(\mathcal{C}, \mathcal{C}_{priv})$*;*
- $\mathsf{send}(\mathcal{C}_{priv}, \mathsf{m}, v, \mathsf{t}) \rightarrow (\xi, \mathsf{aux})$*: the send-message algorithm takes in input the private channel information* $\mathcal{C}_{priv}$*, a message* m *with validity* $v \in \{0,1\}$ *and the turn* $\mathsf{t} < n$*. The algorithm outputs the message correctness proof* ξ *and the channel auxiliary information* aux*.*
- $\mathsf{turntoken}(\mathcal{C}, \mathsf{t}, \{x_0, \ldots, x_{\mathsf{t}-1}\}) \rightarrow x_{\mathsf{t}}$*: this algorithm is executed* at the beginning *of turn* t*. The algorithm parses the channel* $\mathcal{C}$*, the current turn* t *and the set of previously computed turn tokens* $\{x_0, \ldots, x_{\mathsf{t}-1}\}$*, after* Δ *amount of time, the algorithm outputs the turn token* x_{t}*.*
- $\mathsf{valid\text{-}ver}(\mathcal{C}, \mathsf{t}, m, \xi, x_{\mathsf{t}}) \rightarrow \{0,1\}$*:* at the end *of the* t*-th turn, the validity verification takes as input a message* m *and its proof* ξ *and the turn token* x_{t}*. The algorithm outputs the validity* v *for the sent message* m *with proof* ξ*;*
- $\mathsf{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, M_{\mathsf{t}-1}, m, \mathsf{aux}, \xi) \rightarrow \{0,1\}$*:* during *the* t*-th turn, the tamper verification algorithm takes in input the public channel* $\mathcal{C}$*, the current turn* t*, the* ordered block *of messages* $M_{\mathsf{t}-1}$ *which is the list of valid messages for the turn* $\mathsf{t}-1$*, a sent message* m *with proof* ξ *and auxiliary information* aux*. The algorithm verifies if the sent message* m *correctly relates to the previously sent messages contained in the block* $M_{\mathsf{t}-1}$*, thus outputting* 1 *when this is achieved, otherwise* 0*.*
- $\mathsf{ext}(\mathcal{C}, \mathcal{C}_{priv}, \mathsf{t}) \rightarrow x_{\mathsf{t}}$*: the extraction algorithm takes as input the public channel* $\mathcal{C}$*, the private channel* $\mathcal{C}_{priv}$ *and a turn* $\mathsf{t} \leq n$ *and outputs the turn token* x_{t}*, without investing any multiple of* Δ *time;*
- $\mathsf{backward\text{-}ver}(\mathcal{C}, \mathsf{t}, M_{\mathsf{t}-1}, l) \rightarrow \{0,1\}$*: the recovery algorithm takes as input the public channel* $\mathcal{C}$*, the current turn* t*, the previous ordered block* $M_{\mathsf{t}-1}$ *of* $b_{\mathsf{t}-1} = |M_{\mathsf{t}-1}|$ *valid messages* m_i *and an index* $l \in [1, b_{\mathsf{t}-1}]$*. The algorithm outputs if the* l*-th message* $\mathsf{m}^\star$ *in the block* $M_{\mathsf{t}-1}$ *is a correct message for the block* $M_{\mathsf{t}-1}$ at the end *of turn* t*.*

Let us explain how the definition is used to generate a communication channel from Alice P_A to Bob P_B, as depicted in Fig. 57. First, P_A executes setup for an agreed delay Δ and amount of turns n, and obtains the channels $(\mathcal{C}, \mathcal{C}_{priv})$, *e.g.* the public channel $\mathcal{C}$ can consist of P_A's public key and public parameters while the private channel $\mathcal{C}_{priv}$ contains P_A's private key. The knowledge of $\mathcal{C}_{priv}$ allows P_A to quickly compute each turn token x_{t} *directly* as $\mathsf{ext}(\mathcal{C}, \mathcal{C}_{priv}, \mathsf{t})$ while P_B must *sequentially* compute them as $\mathsf{turntoken}(\mathcal{C}, \mathsf{t}, \{x_0, \ldots, x_{\mathsf{t}-1}\})$ and obtain them every Δ amount of time, similarly to a periodic scheduling process. Whenever P_A sends the message m in a turn t, she executes send for a *valid* message in the t *turn* and sends to P_B the tuple $(\mathsf{m}, \xi, \mathsf{aux})$. P_B can execute $\mathsf{valid\text{-}ver}(\mathcal{C}, \mathsf{t}, \mathsf{m}, \xi, x_{\mathsf{t}})$ and verify the message validity only whenever P_B obtains the turn token x_{t}, computable only after $\mathsf{t} \cdot \Delta$ amount of time. This allows P_A to communicate several messages of which P_B cannot immediately verify the validity of m but it has to wait for $\mathsf{turntoken}$ to output the specific turn token x_{t} thus creating the view of turns of the channel.

Message Validity. The sender's inputs are the validity value v, a bit which indicates if the message is considered valid or not, along with the message m itself and the choice of turn t. Only when the turn t ends, the receiver can verify the validity of the message via the $\mathsf{valid\text{-}ver}$ algorithm and the turn token x_{t}.

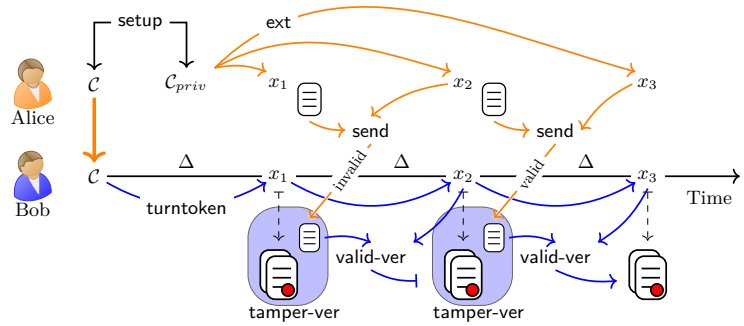

Figure 57: One-Way TBCC scheme usage: Alice submits the public channel $\mathcal{C}$ to Bob, and keeps the private information $\mathcal{C}_{priv}$. On each end of turn, Bob verifies the received messages in order to prevent the addition of invalid messages in the channel.

Definition 44 (Channel Correctness/Message Validity)**.** *Assume a turn* $\mathsf{t} \leq n$ *in a* n*-turn channel generated by the algorithms of Construction 1, then for all message/validity pairs* m *and* v*, the channel is said to be correct if*

$$Pr\left[\mathsf{valid\text{-}ver}(\mathcal{C}, \mathsf{t}, m, \xi, x_{\mathsf{t}}) \neq v \;\middle|\; \begin{array}{l} setup(\lambda, \Delta, n) \rightarrow (\mathcal{C}, \mathcal{C}_{priv}); \\ send(\mathcal{C}_{priv}, m, v, \mathsf{t}) \rightarrow (\xi, aux); \\ ext(\mathcal{C}, \mathcal{C}_{priv}, \mathsf{t}) \rightarrow x_{\mathsf{t}}; \end{array}\right] \leq \mathsf{negl}(\lambda)\,,$$

with probability computed over the random coins of setup*,* send*,* ext *and* valid-ver*.*

Sequentiality and Turn Definition. The turns of the channel rely on the time necessary to compute the token values x_{t} via turntoken, defined in the channel $\mathcal{C}$ during the general setup. Each computed turn-tokens x_{t}, allows the receiver to verify the validity and consistency of all received messages during the turn t, crucially, only at the end of the turn after the expected delay time Δ.

Definition 45 (Sequentiality)**.** *The channel is* Δ*-sequential if for any turn* t*, for any PPT adversary* $\mathcal{A}$ *running in time* $\mu(\mathcal{A}) < \Delta$*, the adversary wins the game* $\mathsf{Game}^{\mathcal{A},\Delta}_{seq}(\lambda, \mathsf{t}, n)$ *of Algorithm 3, with negligible advantage, namely,*

$$\left| Pr[\mathsf{Game}^{\mathcal{A},\Delta}_{seq}(\lambda, \mathsf{t}, n) = 1] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)\,.$$

Algorithm 3 Sequentiality Game $\mathsf{Game}^{\mathcal{A},\Delta}_{seq}(\lambda, \mathsf{t}, n)$ for the adversary $\mathcal{A}$

1: Execute $\mathsf{setup}(\lambda, \Delta, n) \rightarrow (\mathcal{C}, \mathcal{C}_{priv})$;
2: Choose a random message m and validity $v \leftarrow \{0, 1\}$.
3: Execute $\mathsf{ext}(\mathcal{C}, \mathcal{C}_{priv}, i) \rightarrow x_i$ for $i \in [1, \mathsf{t}-1]$ and $\mathsf{send}(\mathcal{C}_{priv}, m, v, \mathsf{t}) \rightarrow (\xi, \mathsf{aux})$
4: $v^* \leftarrow \mathcal{A}\big(\mathcal{C}, \mathsf{t}, m, \xi, \mathsf{aux}, \{x_i\}_{i=1}^{\mathsf{t}-1}\big)$
5: Execute $\mathsf{ext}(\mathcal{C}, \mathcal{C}_{priv}, \mathsf{t}) \rightarrow x_{\mathsf{t}}$
6: If $\mathsf{valid\text{-}ver}(\mathcal{C}, \mathsf{t}, m, \xi, x_{\mathsf{t}}) = v^*$, output 1. Otherwise, 0

Last Turn Tamper Resistance. Given any $\mathsf{t} \leq n$ of a TBCC with public setup information $\mathcal{C}$, define the block $\mathsf{M}_{\mathsf{t}-1}$ as the set of all $j_{\mathsf{t}-1}$ messages in the turn $\mathsf{t}-1$

with respective auxiliary information $\mathsf{aux}_1, \dots, \mathsf{aux}_{j_{\mathsf{t}-1}}$ and sent proof $\xi_1, \dots, \xi_{j_{\mathsf{t}-1}}$. The algorithm $\mathsf{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, \mathsf{M}_{\mathsf{t}-1}, \mathsf{m}, \mathsf{aux}, \xi)$ checks, for any correctly computed message $(\mathsf{m}, \mathsf{aux}, \xi) \in \mathsf{M}_{\mathsf{t}}$, if it correctly relates to the previous turn block $\mathsf{M}_{\mathsf{t}-1}$ by spotting whenever this connection is tampered.

Definition 46 (Last Turn Tamper Resistance)**.** *During the turn $\mathsf{t} \leq n$ of a channel $\mathcal{C}$ between two honest parties with correct message blocks M_i for each turn $1 \leq i < \mathsf{t}$, $\mathcal{C}$ is tamper resistant, if for any PPT adversary $\mathcal{A}$, it holds*

$$Pr\left[\begin{array}{l} \mathit{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, M^{\star}_{\mathsf{t}-1}, m^{\star}, \mathit{aux}^{\star}, \xi^{\star}) = 1 | \\ \quad (M^{\star}_{\mathsf{t}-1}, m^{\star}, \mathit{aux}^{\star}, \xi^{\star}) \leftarrow \mathcal{A}(\mathcal{C}, \mathsf{t}, M_1, \dots, M_{\mathsf{t}-1}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

such that $M^{\star}_{\mathsf{t}-1} \neq M_{\mathsf{t}-1}$ and $\mathit{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, M_{\mathsf{t}-1}, m^{\star}, \mathit{aux}^{\star}, \xi^{\star}) = 1$. The probability is computed over the random coins of $\mathcal{A}$ and algorithm tamper-ver.

Remark 14. *Def. 46 is strictly dependent on the current turn being t, in the sense that it does not deal directly with tampering of messages in earlier turns than $\mathsf{t}-1$. By considering the definition for $1 \leq \mathsf{t} \leq n$, it covers all the turns for the channel,* ***except*** *the n-th turn creating the* ***last message tamper*** *that will not be possible to verify because there are* "no more turns", i.e. *the computation $\mathit{turntoken}(\mathcal{C}, n+1, \{x_i\}_{i=0}^{n})$ is not defined or the result must not be used.*

Communication Consistency. For any turn $\mathsf{t} \leq n$ of a one-way channel $\mathcal{C}$, the channel is *consistent until turn* $\mathsf{t}-1$ whenever the *valid* messages view between the parties is the same during the turn t, *i.e.* an adversary **must not** be able to force a wrong message history, regardless if it is the sender or the receiver.

Definition 47 (Consistency)**.** *During turn $\mathsf{t} \leq n$ of a one-way TBCC channel $\mathcal{C}$ between two parties with correct message blocks M_i for each turn $1 \leq i < \mathsf{t}$, the channel is* ***consistent until turn*** *$\mathsf{t}-1$, if for any PPT adversary $\mathcal{A}$, it holds*

$$\begin{array}{l} Pr[\mathit{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, M^{\star}_{\mathsf{t}-1}, m^{\star}, \mathit{aux}^{\star}, \xi^{\star}) = 1 | \\ \quad (M^{\star}_{\mathsf{t}-1}, m^{\star}, \mathit{aux}^{\star}, \xi^{\star}) \leftarrow \mathcal{A}(\mathcal{C}, \mathsf{t}, M_1, \dots, M_{\mathsf{t}-1}) \ ] \leq \mathsf{negl}(\lambda) \end{array}$$

such that $M^{\star}_{\mathsf{t}-1} \neq M_{\mathsf{t}-1}$, $\mathit{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, M_{\mathsf{t}-1}, m^{\star}, \mathit{aux}^{\star}, \xi^{\star}) = 1$ and for all the messages of the tampered block, along with auxiliary information and proof, i.e. $(m^{\star}_{j_i}, \mathit{aux}^{\star}_{j_i}, \xi^{\star}_{j_i}) \in M^{\star}_{\mathsf{t}-1}$, *it holds* $\mathsf{valid\text{-}ver}(\mathcal{C}, \mathsf{t}-1, m^{\star}_{j_i}, \xi^{\star}_{j_i}, x_{\mathsf{t}-1}) = 1$ *The probability is computed over the random coins of $\mathcal{A}$, tamper-ver and* valid-ver*.*

3.2 One-Way Channel Instantiation.

Let $\Delta \in \mathbb{R}_+$ be a time-delay and $n \in \mathbb{N}$ a maximal turn number, both chosen by Alice, denoted with P_A. Let H and $\overline{\mathsf{H}}$ be respectively regular and Δ-delay hash functions. Let $(\mathsf{GenPuz}, \mathsf{SolPuz})$ be the $(n\Delta)$-TLP of Def. 42 based on $\overline{\mathsf{H}}$.

Construction 1. *Let λ be the security parameter, $n \in \mathbb{N}$ number of turns, a sender P_A and a receiver P_B. Instantiate the one-way channel scheme with the PPT algorithms $(\mathit{setup}, \mathit{send}, \mathit{ext}, \mathit{turntoken}, \mathsf{valid\text{-}ver}, \mathit{tamper\text{-}ver})$ defined as:*

- $\mathit{setup}(\lambda, \Delta, n) \rightarrow (\mathcal{C}, \mathcal{C}_{priv})$*: to setup the communication channel, P_A parses the security parameter λ, the delay Δ and the number of turns n and executes the algorithm $\mathsf{GenPuz}(\lambda, (n, \Delta))$ as defined in Def. 42 and obtains the n turn puzzle with solution $(\mathbf{y}, \pi)$. Output $(\mathcal{C}, \mathcal{C}_{priv})$ as $(\mathbf{y}, \pi)$;*
- $\mathit{send}(\mathcal{C}_{priv}, m, v, \mathsf{t}) \rightarrow (\xi, \mathit{aux})$*: to send a message m with validity v in the turn $\mathsf{t} < n$, P_A parses the private channel information $\mathcal{C}_{priv} = \pi$, and compute the values $h_{\mathsf{t}-1} \coloneqq H(M_{\mathsf{t}-1}, m, \pi_{\mathsf{t}-1})$, $\xi \coloneqq H(m, \pi_{\mathsf{t}})$ and $\sigma \coloneqq H(m, \xi, \pi_{\mathsf{t}+1})$ where $M_{\mathsf{t}-1}$*

is the ordered list of valid messages in the turn $(\mathsf{t}-1)$*, together with validity proof and auxiliary information. The sending algorithm outputs, if* $v=1$*, the message correctness proof* ξ *and the channel auxiliary information* $\mathsf{aux}=(h_{\mathsf{t}-1},\sigma)$*, otherwise random values* (ξ,aux) *different from the correct ones.*

- $\mathsf{turntoken}(\mathcal{C},\mathsf{t},\{x_0,\ldots,x_{\mathsf{t}-1}\})\rightarrow x_{\mathsf{t}}$*: this algorithm is executed by the receiver* P_B *at the beginning of turn* t*. It parses the channel* $\mathcal{C}=\mathbf{y}$ *and continually executes* $\mathsf{SolPuz}(\mathbf{y})$ *by considering that every* $\pi_i:=x_i$ *for the* t *partial solution. After* Δ *amount of time, the output of the algorithm is* $x_{\mathsf{t}}:=\pi_{\mathsf{t}}$*.*
- $\mathsf{valid\text{-}ver}(\mathcal{C},\mathsf{t},m,\xi,x_{\mathsf{t}})\rightarrow\{0,1\}$*: at the end of the* t*-th turn, the validity verification takes as input a message* m *and its proof* ξ *and the turn token* $x_{\mathsf{t}}=\pi_{\mathsf{t}}$*. Output* 1 *if the equality* $H(m,\pi_{\mathsf{t}})\stackrel{?}{=}\xi$ *holds. Otherwise,* 0*;*
- $\mathsf{tamper\text{-}ver}(\mathcal{C},\mathsf{t},M_{\mathsf{t}-1},m,\mathsf{aux},\xi)\rightarrow\{0,1\}$*: during the* t*-th turn, the receiver* P_B *verify the correctness of the ordered* $(\mathsf{t}-1)$*-th block* $M_{\mathsf{t}-1}$ *which contains the previously valid ordered messages* $\{m_i\}_{i=1}^{j_{\mathsf{t}-1}}$ *for some* $j_{\mathsf{t}-1}\in\mathbb{N}$*, by parsing the auxiliary information as* $\mathsf{aux}=(h_{\mathsf{t}-1},\sigma)$ *and outputs the result of the equality verification* $H(M_{\mathsf{t}-1},m,\pi_{\mathsf{t}-1})\stackrel{?}{=}h_{\mathsf{t}-1}$*.*
- $\mathsf{ext}(\mathcal{C},\mathcal{C}_{priv},\mathsf{t})\rightarrow x_{\mathsf{t}}$*: the extraction algorithm takes as input the public channel* $\mathcal{C}$*, the private channel* $\mathcal{C}_{priv}=\pi$ *and a turn* $\mathsf{t}\leq n$ *and outputs* $x_{\mathsf{t}}=\pi_{\mathsf{t}}$*;*
- $\mathsf{backward\text{-}ver}(\mathcal{C},\mathsf{t},M_{\mathsf{t}-1},l)\rightarrow\{0,1\}$*: the algorithm takes as input the public channel* $\mathcal{C}$*, the current turn* t*, the previous ordered block* $M_{\mathsf{t}-1}$*, of accepted message* m_i *for* $i\in[1,j_{\mathsf{t}-1}]$*, and an index* l *such that* $m^\star$ *is the* l*-th message in the block* $m^\star=m_l\in M_{\mathsf{t}-1}$ *with auxiliary information* $\mathsf{aux}^\star=\mathsf{aux}_l=(h_{\mathsf{t}-2}{}^\star,\sigma^\star)$*.* $\mathsf{backward\text{-}ver}$ *computes* $\xi^\star=H(m^\star,\pi_{\mathsf{t}-1})$ *and outputs if* $H(m^\star,\xi^\star,\pi_{\mathsf{t}})\stackrel{?}{=}\sigma^\star$*. The* $\mathsf{backward\text{-}ver}$ *algorithm verifies at the end of turn* t *if the message* $m^\star$ *is a correct message for the block* $M_{\mathsf{t}-1}$*.*

Proposition 11. *The proposed one-way channel instantiation of Construction 1 achieves channel correctness as stated in Def. 44.*

Proof. Consider a turn $\mathsf{t}\leq n$ for an n-turn one-way channel defined by executing $(\mathcal{C},\mathcal{C}_{priv})\leftarrow\mathsf{setup}(\lambda,\Delta,n)$. For any message m with validity v, compute the value $\mathsf{send}(\mathcal{C}_{priv},\mathsf{m},v,\mathsf{t})\rightarrow(\xi,(\mathsf{h}_{\mathsf{t}-1},\sigma))$ of which ξ is either $\mathsf{H}(\mathsf{m},\pi_{\mathsf{t}})$ if v=1 otherwise it is an incorrect value. Furthermore execute $\mathsf{ext}(\mathcal{C},\mathcal{C}_{priv},\mathsf{t})\rightarrow\pi_{\mathsf{t}}$. By definition, we have that $\mathsf{valid\text{-}ver}(\mathcal{C},\mathsf{t},\mathsf{m},\xi,\pi_{\mathsf{t}})$ outputs as validity the equality of $\mathsf{H}(\mathsf{m},\pi_{\mathsf{t}})\stackrel{?}{=}\xi$ which is 1, when correctly computed, and 0 otherwise. Assume the existence of an adversary $\mathcal{A}$ able to break the correctness property with some non-negligible probability $\nu>0$, *i.e.* $\mathcal{A}$ is able to produce an invalid pair $(\mathsf{m}^\star,\pi_{\mathsf{t}}{}^\star)$ such that $\mathsf{valid\text{-}ver}(\mathcal{C},\mathsf{t},\mathsf{m}^\star,\xi,\pi_{\mathsf{t}}{}^\star)=1$ for some given digest ξ with probability ν. Let $\epsilon_{\mathsf{H}.pre}$ be the assumed negligible probability of finding a digest pre-image for H of ξ. Construct an adversary $\mathcal{B}$ that reduce the pre-image computation to the one-way correctness by querying $\mathcal{A}$ for a pair $(\mathsf{m}^\star,\pi_{\mathsf{t}}{}^\star)$ for the digest ξ. $\mathcal{B}$ outputs as pre-image the value $(\mathsf{m}^\star,\pi_{\mathsf{t}}{}^\star)$. We conclude that:

$$\nu=\mathsf{Pr}\left[\mathsf{valid\text{-}ver}(\mathcal{C},\mathsf{t},\mathsf{m},\xi,x_{\mathsf{t}})\neq v\;\middle|\;\begin{array}{l}\mathsf{setup}(\lambda,\Delta,n)\rightarrow(\mathcal{C},\mathcal{C}_{priv});\\ \mathsf{send}(\mathcal{C}_{priv},\mathsf{m},v,\mathsf{t})\rightarrow(\xi,\mathsf{aux});\\ \mathsf{ext}(\mathcal{C},\mathcal{C}_{priv},\mathsf{t})\rightarrow x_{\mathsf{t}};\end{array}\right]\leq\epsilon_{\mathsf{H}.pre}$$

which is absurd. Thus proving the correctness property. □

Proposition 12. *The proposed one-way channel instantiation of Construction 1 achieves sequentiality as stated in Def. 45.*

Proof. Consider the sequentiality game $\mathsf{Game}_{seq}^{\mathcal{A},\Delta}(\lambda,\mathsf{t},n)$ in which the challenger generates the communication channel $(\mathcal{C},\mathcal{C}_{priv})$ and let $\mathsf{t}\leq n$ be an arbitrary turn in which

the adversary is challenged. The challenger chooses an arbitrary message m and validity $v \leftarrow \{0,1\}$ and executes $\mathsf{send}(\mathcal{C}_{priv}, \mathsf{m}, v, \mathsf{t}) \rightarrow (\xi, \mathsf{aux})$. The adversary $\mathcal{A}$ wins the game if the output $v^* \leftarrow \mathcal{A}\big(\mathcal{C}, \mathsf{t}, m, \xi, \mathsf{aux}, \{x_i\}_{i=1}^{\mathsf{t}-1}\big)$ is the challenger's chosen validity v **and** the execution time for the adversary is bounded as $\mu(\mathcal{A}) < \Delta$. $\mathcal{A}$ can therefore be used by an adversary $\mathcal{B}$ to reduce the Δ-delay property for the Δ-delay hash function to the one-way sequentiality game. Briefly, if we assume $\mathcal{A}$ to have a non-negligible probability to compute v, $\mathcal{B}$ is able to break the Δ-delay property which is assumed to be hard. □

Proposition 13. *The proposed one-way channel instantiation of Construction 1 achieves last turn tamper resistance as stated in Def. 46.*

Proof. Consider a communication between two honest parties to generate the blocks M_i for $i \in \{1, \ldots, t-1\}$ where $t \leq n$ is the turn in which the adversary $\mathcal{A}$ will output the tuple $(\mathsf{M}^\star, \mathsf{m}^\star, (\mathsf{h}^\star, \sigma^\star), \xi^\star)$, which contains a tampered block for the turn $t-1$, a tampered message and the related auxiliary information and the tampered validity proof. Observe that the verification algorithm will compute $\mathsf{tamper\text{-}ver}(\mathcal{C}, t, \mathsf{M}_{\mathsf{t}-1}, \mathsf{m}^\star, (\mathsf{h}^\star, \sigma^\star), \xi^\star)$ with the correct block, which will verify the equality of $\mathsf{H}(\mathsf{M}_{t-1}, \mathsf{m}^\star, \pi_{\mathsf{t}-1}) \stackrel{?}{=} \mathsf{h}^\star$. Obviously, $\mathcal{A}$ can always generate, for any messages, correctly evaluated digests. However, in order to correctly consider it a tamper, the adversarial tamper must verify the algorithm with the tampered block. Then, to allow the existence of two correct but different block visions, *i.e.* formally $\mathsf{tamper\text{-}ver}(\mathcal{C}, t, \mathsf{M}^\star, \mathsf{m}^\star, (\mathsf{h}^\star, \sigma^\star), \xi^\star)$, which is equivalent to $\mathsf{H}(\mathsf{M}^\star, \mathsf{m}^\star, \pi_{\mathsf{t}-1}) \stackrel{?}{=} \mathsf{h}^\star$. Assume by absurd that such $\mathcal{A}$ exists and outputs correct tampers with non-negligible probability $\nu > 0$. Intuitively, construct an adversary $\mathcal{B}$ that reduce the second pre-image computation to the one-way tampering by querying $\mathcal{A}$. $\mathcal{A}$ must provide a second pre-image $(\mathsf{M}^\star, \mathsf{m}^\star)$ of the digest $\mathsf{h}^\star$ obtained from $(\mathsf{M}_{t-1}, \mathsf{m}^\star)$. Thus, $\mathcal{B}$ outputs a second pre-image of $\mathsf{h}^\star$ with probability $\nu \leq \epsilon_{\mathrm{H}.2pre}$ which is assumed to be negligible. □

Proposition 14. *Consistency ⇔ last turn tamper resistant and correctness.*

Proof. The proof of this proposition is trivial. Our definition of consistency is similar to the definition of tamper resistance where we additionally require the tampered block to be formed only by *correct* messages. Therefore, a consistent channel is trivially correct and tamper resistant. For the opposite implication, assume that the channel is non-consistent, *i.e.* an adversary can compute a wrong message view in a specific turn. This is true if and only if the adversary can create a correct tamper block which contains at least a wrong message-proof ξ and auxiliary information tuple aux. This implies that a non-consistent channel allows to break the correctness and tamper resistance property. □

3.3 Two-Way TBCC

In this section, we instantiate a two-way TBCC and explain how to correctly realise the *recovery procedure*, *i.e.* a procedure executed between the parties that allows them to force the communication's correctness and coherence.

Consider the parties P_A and P_B and let both independently setup the consistent one-way channel of Construction 1 which casts them both as receiver and sender into two independent channels each. Both parties can send a message to the other one in the channel they created. Concurrently, each party keeps track of its local turn to receive and check messages by *(1)* continuously executing $\mathsf{turntoken}$ and *(2)* keeping of the previously generated turn tokens x_i for $i \leq \mathsf{t}$.

Protocol 3 (The Two-Way TBCC Protocol)**.** *Given two parties P_A and P_B, an integer value n and real non-zero value Δ, define the (Two-Way) TBCC across n turns with delay Δ with the procedures:*

- **Setup:** *on input the security parameter λ, P_A (respectively P_B) executes the algorithm $\mathsf{setup}(\lambda, \Delta, n)$, obtains $(\mathcal{C}_A, \mathcal{C}_{A,priv})$, and sends $\mathcal{C}_A$ to P_B, which replies with $\mathcal{C}_B$. P_A outputs the two-way TBCC channel information $(\mathcal{C}_A, \mathcal{C}_B)$, along with its respective private information $\mathcal{C}_{priv}$ and P_A performs $\mathsf{turntoken}(\mathcal{C}_B, 1, x_{B,0})$;*
- **Local Turn (analogously for P_B):** *on receiving a call to this procedure, P_A returns the current local turn t corresponding to the last computed $x_{P_B,\mathsf{t}}$;*
- **Send Message (analogously for P_B):** *on a given local turn t, when P_A receives the input (m, v), it executes $\mathsf{send}(\mathcal{C}_{A,priv}, \mathsf{m}, v, \mathsf{t}) \rightarrow (\xi, \mathsf{aux})$ where the previous block digest is computed as $\mathsf{h}_{\mathsf{t}-1} \coloneqq \mathsf{H}(\mathsf{M}_{\mathsf{t}-1}, \mathsf{m}, \pi^{P_A}_{\mathsf{t}-1}, \pi^{P_B}_{\mathsf{t}-1})$, and sends $(\mathsf{m}, \xi, \mathsf{aux})$ to P_B;*
- **Reveal Validity (analogously for P_A):** *at the end of the local turn t,* i.e. *when the algorithm $\mathsf{turntoken}(\mathcal{C}_A, \mathsf{t}, \{x_{A,0}, \dots, x_{A,\mathsf{t}-1}\})$ outputs the token $x_{A,\mathsf{t}}$, P_B executes $\mathsf{valid\text{-}ver}(\mathcal{C}_A, \mathsf{t}, m_i, \xi_i, x_{A,\mathsf{t}}) \rightarrow v_i$, and outputs the block of* both the parties *valid messages $\mathsf{M}_{\mathsf{t}}=\{(m_i, \xi_i, \mathsf{aux}_i)\}_i$ along with the turn token t whenever $v_i=1$. Furthermore, for all the messages m_i, $\mathsf{tamper\text{-}ver}(\mathcal{C}_A, \mathsf{t}, \mathsf{M}_{\mathsf{t}-1}, \mathsf{m}_i, \mathsf{aux}_i, \xi_i)$ is executed and if any result is 0, abort the communication. If $\mathsf{t}+1 > n$, then output* CLOSE *and stop. Otherwise, execute $\mathsf{turntoken}(\mathcal{C}_A, \mathsf{t}+1, \{x_{A,0}, \dots, x_{A,\mathsf{t}}\})$.*

Remark 15. *The TBCC protocol naturally extends the one-way properties of correctness and tamper resistance to the two-way channel. For example, if the two-way channel is tamperable, it means the adversary can tamper at least one direction of the communication channel. In other words, tamper the one-way channel.* Mutatis mutandis *the same is true for the correctness property.*

Turn Synchronization and Consistency. When considering the two-way protocol by instantiating two one-way turn based schemes, an additional problem that naturally arises is *turn synchronization between the parties*. Consider the parties P_A and P_B communicating using Proto. 3 which depends on the specific one-way channels $\mathcal{C}_A$ and $\mathcal{C}_B$. The specific channel turn is identified by the input of the algorithm turntoken which are, *almost surely*, never synchronized, *i.e.* the outputs are disclosed in different moments. This timing lack creates a problem in which a message m might be seen in turn t by P_A and in turn $(\mathsf{t}+1)$ by P_B. We capture this idea by formalizing the *turn synchronization* property.

Definition 48 (Turn Synchronization)**.** *Let P_A and P_B be parties communicating over the two-way TBCC. The TBCC channel $(\mathcal{C}_A, \mathcal{C}_B)$ is **turn-consistent** if both players have a **unique and equal** way to decide in which turn the message m belongs even then the **local turns** of the two parties are different.*

The TBCC without turn synchronization cannot achieve communication consistency since the parties might disagree in which block M the message m belongs, making it unlikely to create an unique communication history. Intuitively, achieving sequentiality means that the turntoken algorithm is defining a *"clock"*, *i.e.* sequential *"ticks"* distanced by some amount of time, while being desynchronized means that the parties have *"different clocks"* where one of the two is always *"late"*. We prove that if we have a sequential one-way scheme, then there exists a natural way to achieve turn-consistency by letting the parties **avoid communicating** in between the *"ticks"* thus allowing the *"late clock"* to sync.

Proposition 15. *Let P_A and P_B be parties communicating via the two-way TBCC protocol, constructed from a sequential one-way scheme as in Def. 45. The strategy*

of (i) dropping communicated messages during de-synchronization, i.e. *the local turn between the parties is different; and (ii)* globally *advance the turn whenever both parties have the same local turn; allows turn-consistency as in Def. 48.*

Recovery Procedure. We consider the existence of a *recovery procedure* that should be executed whenever a party spots a possible communication tamper and, instead of directly aborting the protocol, the two parties try to find a common correct message block. In other words, the algorithm tamper-ver from Construction 1 takes as input the last block views M_{P_A} and M_{P_B} that the two parties have and either outputs a commonly agreeable block M or aborts.

Definition 49 (Recovery)**.** *Define the recovery procedure for Proto. 3 as the procedure executed during turn* $\mathsf{t} \leq n$ *by* P_A *(resp.* P_B*) whenever the tamper verification* $\mathsf{tamper\text{-}ver}(\mathcal{C}, \mathsf{t}, M_{\mathsf{t}-1}, m, aux, \xi)$ *is equal* 0 *and defined as:*

- **Recovery***:* P_A *sends its view* $M^A_{\mathsf{t}-1}$ *to* P_B *from whom it receives the view* $M^B_{\mathsf{t}-1}$ *which is a ordered list of messages* $\{m_i\}_{i=1}^{j_{\mathsf{t}-1}}$ *and, additionally, for every message the received auxiliary information* σ*. After identifying the set of indexes* I *where the views differ, for each index* $l \in I$*, if the message* m_l *is a message from* P_B*, then* P_A *executes* $\mathsf{backward\text{-}ver}(\mathcal{C}_B, \mathsf{t}, M^B_{\mathsf{t}-1}, l)$*, otherwise* P_B *will compute* $\mathsf{backward\text{-}ver}(\mathcal{C}_A, \mathsf{t}, M^A_{\mathsf{t}-1}, l)$*. Either the case, if the result is* 1*, both parties are forced to use the message* m_l *resolving the discrepancy and saving the result into the same resolved block* $M_{\mathsf{t}-1}$*. Otherwise, if there exists an index for which the result is* 0*, the communication is aborted.*

The spirit of the TBCC is *"if anything seems wrong, abort!"*. This forces the parties to behave honestly otherwise nothing can be achieved, meaning there can never exist **two** different correct views. During the recovery procedure, the communication is paused and completely verified and fixed before continuing and, if necessary, aborted because it is unrecoverable. The receiver must be aware and promptly alert the sender if h_{i-1} is wrong and, if it is the case, only the receiver can force the sender to adopt a specific message m_i by exhibiting the received proof σ_i, only computable by the sender.

Formally, suppose P_A and P_B are correctly communicating until the i-th turn, *i.e.* all the blocks until M_{i-1} are consistent. P_A sends $\left(\mathsf{h}^A_{i-1}, \mathsf{m}^A_i, \xi^A_i, \sigma^A_i\right)$ and P_B does the same with the message m^B_i. Let us suppose that the values $\{\xi^A_i, \xi^B_i\}$ are correct otherwise the messages will be discarded by valid-ver. Thus the correct next block is $\mathsf{M}_i = \{\mathsf{m}^A_i, \mathsf{m}^B_i\}$. Whenever the turn $(i+1)$ starts, P_B and P_A must share the block digests h^A_i and h^B_i and suppose they are not equal.

The recovery procedure is executed and P_B will publish the block-view $\{\mathsf{m}^A_i, \mathsf{m}^B_i\}$, respectively P_A must do the same, and there must be at least a different message pair, *w.l.o.g.* suppose it is message m^A_i and $\mathsf{m}^{A^\star}$. Since this is the message that Alice sent, in the recovery, we will just consider Bob's view $\mathsf{m}^{A^\star}$ with received auxiliary information $\sigma^{A^\star}$ which Bob cannot correctly forge by assumption, *i.e.* he cannot produce a correct valid pair. Therefore P_B can only re-publish what P_A sent **or** abort the communication. Regardless of P_B's maliciousness, he is unable to modify Alice's messages and therefore the procedure continues only if $\sigma^{A^\star}$ is correctly computed by P_A. In the case that Bob's message m^B_i is different, Alice's vision is considered. If Bob is honest, the previous discussion applies for Alice. Otherwise, Bob might try to force the acceptance of a different pair $(\mathsf{m}^{B^\star}, \sigma^{B^\star})$. Since his vision during recovery is not considered, he must have sent the tampered values $(\mathsf{m}^{B^\star}, \sigma^{B^\star})$ *before* but if this is the case, either Alice is presenting the tampered pair $(\mathsf{m}^{B^\star}, \sigma^{B^\star})$, which makes the pair not longer a tamper, since it is correctly received by Alice and not later modified, or by sending an incorrect pair that will lead to aborting the communication. *Mutatis mutandis*, the same is

true when switching P_A and P_B roles. If everything is correct, the block vision is consolidated, communication can resume and the only real cost is that both P_A and P_B lost a single turn.

4 Collectively Flipping Coins over the TBCC

In this section, we sketch a protocol that allows two parties to *collectively flip a coin* which allows them to commonly create a random string. Our TBCC protocol is constructed from time-lock puzzles which are used in similar applications, as:

- a user can create encrypted time capsule, *i.e.* an encrypted message that is meant to only be decryptable after a designed amount of time;
- a user can provide a signature that can only be verified in the future.

As discussed by Rivest *et al.* [RSW96], these are founded on the concept of releasing a *timed commitment* that can be decommitted after a specific amount of time.

The provided coin-flip solution is *simplistic* and it has the main goal of showing the TBCC's expressiveness/potentiality. To provide a formal security analysis, TBCC must be proven secure against active adversaries and general protocol's composability which, as previously assumed, are left open for future research.

Flipping Coins over TBCC.

The underlying idea is that two parties, communicating over a TBCC's instance, are able to *jointly* flip a coin by both time-committing to some randomness which is later revealed and used to compute the coin result. By repeatedly flipping coins, the results produce a random string which is guaranteed to be consistent since communicated over TBCC.

Let us provide a formalisation of the collectively coin-flip between Alice P_A and Bob P_B. These protocols are defined by a set of choices Σ and a set of rules that allows to determine the *result* between any two choices, denoted with the function $\mu(\cdot,\cdot)$. Formally, the collective coin-flip protocol is therefore defined as:

(a) P_A and P_B set up the two-way TBCC protocol of Proto. 3 and obtain the public channel $\mathcal{C} = (\mathcal{C}_{P_A}, \mathcal{C}_{P_B})$;
(b) In the current turn, P_A selects its choice $a \in \Sigma$ and sends on $\mathcal{C}$ as a valid message, *i.e.* P_A execute the sending procedure with the message $(a, 1)$. For each other choice $a^\star \in \Sigma$, P_A sends the non-valid message $(a^\star, 0)$. Respectively, P_B sends his valid and invalid messages;
(c) At the end of the turn, P_A computes the validity of P_B's received messages and obtains b. Respectively for P_B;
(d) Both the parties compute $\mu(a, b)$ and, if necessary, repeat the game. If the channel loses consistency, *i.e.* one of the party tries to tamper the results, the communication is aborted;
(e) The random string is obtained by concatenating several consecutive results of the consistent channel.

The *"commit-decommit"* phase created by the turn token is key to allow a fair-play since, for example, if P_A knows P_B's choice b in advance, she can select a winning choice $a^\star$. Furthermore, ϕ must be defined even in the case of one party not participating in the round or it tries to cheat by proposing multiple choices. We are now left to define the choice's set Σ and the rule's map $\mu(\cdot,\cdot)$. Σ contains the choices head and tail, respectively 1 and 0 and, additionally, a special element x that represents any non-correct choice, *i.e.* a party does not correctly participate in the game. Define the map μ as $\mu(a, b) = a \oplus b$, *i.e.* the xor between the inputs where the special element is mapped as $\mu(\mathsf{x}, a) = \mu(a, \mathsf{x}) = a$ for each $a \in \Sigma$ and we consider a special state X used to denote that both player wrongly participated in the flipping, *i.e.* $\mu(\mathsf{x}, \mathsf{x}) = \mathsf{X}$. In a nutshell, $\mu(a, b)$ computes the xor of both the parties inputs whenever they are correctly participating in

the coin-flip. Complementary, if both the parties wrongly flip the coin, $\mu(\mathsf{x}, \mathsf{x})$ returns that the coin is in a *"draw position"* with *"no winner"*. Whenever a party, *e.g.* P_A, wrongly participates in the protocol, $\mu(\mathsf{x}, b)$ awards the other party P_B for correctly behaving and let P_B's choice be the final result. This forces the parties to correctly behave to avoid the other party highly influence the coin-flip. For example, suppose that P_A selects 1 as her first choice and sends to P_B the TBCC messages $(1, 1)$ and $(0, 0)$ during the current turn. By the sequentiality property, P_B is unable to discover *"which message is the valid one"* and therefore has no advantage and must therefore provide his own choice, *w.l.o.g.* let P_B choose 0. At the end of the turn, the valid messages are maintained thus the block will contain P_A's message 1 and P_B's one 0. Both the parties can now compute $\mu(1, 0) = 1$ and acknowledge that the coin flip is 1. The TBCC protocol guarantees communication coherence which implies that, whenever repeating the game, both the parties **must** accept the previous communication transcription. In other words, *while* communicating over $\mathcal{C}$, P_A and P_B cannot modify the output of the different rounds played. This means that if the result is 1, in the next round P_B cannot pretend a different outcome and must accept it if he wants to participate in the next round. The game output's transcript can be seen as a random string between P_A and P_B which cannot be tampered with by a malicious adversary. Additionally, every time the adversary is caught tampering or deny the communication, the whole protocol is terminated making it impossible for the adversary to gain any relevant advantage.

We must point out that our protocol **does not** approximate a *public coin flip* one which can be used to generate the common reference string model. In the public coin-flip protocol, the two parties obtain a random coin-flip *without* introducing their own personally sampled randomness. For this reason, our protocol can be used to approximate an empirical version of the common reference string model in which the parties *actively collaborate* to sample a random string.